Synthetic Organisms and Self-Designing Systems*

W. B. Dress

Instrumentation and Controls Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831-6007

CONF-8905130--1

DE89 010318

Abstract

This paper examines the need for complex, adaptive solutions to certain types of complex problems typified by the Strategic Defense System and NASA's Space Station and Mars Rover. Since natural systems have evolved with capabilities of intelligent behavior in complex, dynamic situations, it is proposed that biological principles be identified and abstracted for application to certain problems now facing industry, defense, and space exploration.

Two classes of artificial neural networks are presented—a non-adaptive network used as a genetically determined "retina," and a frequency-coded network as an adaptive "brain." The role of a specific environment coupled with a system of artificial neural networks having simulated sensors and effectors is seen as an ecosystem. Evolution of synthetic organisms within this ecosystem provides a powerful optimization methodology for creating intelligent systems able to function successfully in any desired environment.

A complex software system involving a simulation of an environment and a program designed to cope with that environment are presented. Reliance on adaptive systems, as found in nature, is only part of the proposed answer, though an essential one. The second part of the proposed method makes use of an additional biological metaphor—that of natural selection—to solve the dynamic optimization problems that every intelligent system eventually faces. A third area of concern in developing an adaptive, intelligent system is that of real-time computing. It is recognized that many of the problems now being explored in this area have their parallels in biological organisms, and many of the performance issues facing artificial neural networks may find resolution in the methodology of real-time computing.

*Research performed at Oak Ridge National Laboratory, operated by Martin Marietta Energy Systems, Inc., for the U.S. Department of Energy under Contract No. DE-AC05-84OR21400.

Introduction

The Strategic Defense System is archetypical of a certain class of complex problems that are becoming increasingly important to defense and industry as the 21st century nears. Additional examples of such problems include optimized control of nuclear power plant clusters, design of new and specific molecular medicines, managing the space station, and controlling unmanned planetary exploration vehicles. The complexity of these and similar problems raises serious questions concerning the usual methodology of hardware and software design and makes impossible demands on current methods of reliability testing and system verification. The present approach in treating complex systems is to create a simulation presumed to be representative of the actual system in its essential details. Studying a simulation is thought to be a practical alternative to reality when issues of complexity, prohibitive expense, and impossibility of adequate testing are concerned.

The need for an accurately detailed description of the physical system components and their interactions becomes paramount, as the behavior of the simulation is the basis for developing strategies to cope with real systems. This points to what may be a major flaw in current software simulation and modeling philosophy, as any change requires extensive reprogramming of major parts of the entire simulation. Thus, the predictive power of a simulation to be used for the design of a complex system is easily compromised.

Incompleteness of information concerning the real, physical systems being simulated imposes another intolerable burden on the simulations and support teams. In addition to the reliance demanded from implementation hardware (sensors, communications, effectors, and processors), we are demanding that programmers perform flawlessly under extreme stress of time constraints and imperfect knowledge. This is clearly unacceptable, as anyone who has ever attempted to write, debug, and run even the simplest program can attest.

This paper results from a search for a methodology to attack these very real issues of hardware and software complexity, reliability, and dynamic variability; and to show how software systems might become self-designing, overcoming both the severe constraints noted above and providing the confidence essential to deployment by ensuring reliable and correct functioning in a changing environment. The ideas presented below are still in their infancy, but they have been partially tested with encouraging results. The main research effort is to determine which principles to abstract from nature, the extent of abstraction necessary, and the details required for creating intelligent machines; for it is only through adaptive, intelligent systems that the limitations noted above can be overcome.

The problems of modeling and simulation are discussed first, and a new principle of software engineering is proposed. The question of whether a complex system can be simulated is raised. Examination of issues leads to a proposal for creating synthetic organisms to solve certain complex problems. Two network models are presented as a vehicle for implementing a self-designing, complex system. Results of the two evolutionary programs based on these networks are discussed, and a number of possible extensions to the methods are given.

Simulation & Modeling

For problems of sufficient complexity, a step-by-step simulation is the most efficient means of obtaining predictions of system behavior. Wolfram[1] has argued that the behavior of certain systems may be effectively found only by an explicit, step-by-step simulation, and he considers such systems to be "computationally irreducible." Wolfram's argument amounts to showing a contradiction between the assumptions of a universal computer for such calculations and the existence of an algorithmic shortcut for the simulation. Physics and engineering are concerned primarily with the computationally reducible, while most biological systems are computationally irreducible. For example, "the development of an organism from its genetic code" may well belong to the latter class.[1] Wolfram goes on to suggest that "the only way to find out the overall characteristics of the organism from its genetic code may be to grow it explicitly. This would make large-scale computer-aided design of biological organisms, or 'biological engineering,' effectively impossible: only explicit search methods analogous to Darwinian evolution could be used." [1]

Given the dynamic nature of a complex, real-time control problem, the phrase "biological organism" may be replaced with "software" and "biological engineering" with "software engineering," extending the range of applicability of the previous sentence. Wolfram's suggestion then becomes a new principle of software engineering for truly complex problems. It is this principle that we wish to explore along with neural networks and adaptive systems.

Complex Systems: Where Simulations Fail

Why are we concerned with biology and problems of computational irreducibility? Artificial neural networks are well-understood computational structures firmly rooted in the mathematics of systems of first-order

differential equations, and their proponents claim that many pressing problems will yield to the new paradigm of computational neuroscience. On the other hand, a biological system is one that lives in and has been optimized for a certain dynamic ecosystem. Such systems are complex and not well understood from the simple-system perspective. Evidence is accumulating from many quarters that systems combining information management and real-time control of complicated hardware are likewise not simple. By the very nature of an algorithm, algorithmic methodologies developed to cope with simple systems will most assuredly fail when applied to these complex problems. Indeed, it is already evident that expert systems (deterministic decision trees) become brittle when the application domain is slightly altered, as does any algorithmic structure when used outside its range of applicability. Note that the *ad hoc* addition of "fuzzy reasoning" by adding Bayesian logic or fuzzy sets does not cure this problem: once the ranges of variation are specified, the system is still essentially deterministic. Although simulation may well be a practical way to study certain problems, it is too much to hope that the limitations imposed by brittle programs and inadequate knowledge can be overcome by a purposefully designed simulation.

To go beyond simulation to a synthetic organism that solves a complex, real-time problem in an effective manner is seen as the next logical development in computer science. A simple system, by definition, is one allowing separation between states and dynamical laws, i.e., the intrinsic nature of the system and its response to external effects.[2] The author of this view of physical systems, Robert Rosen, suggests that "any system for which such a description cannot be provided ... [is] *complex*. Thus, in a complex system, the causal categories become intertwined, in such a way that no dualistic language of states plus dynamical laws can completely describe it."[2] Computability of complex systems may be examined by considering the well-known Church-Turing Thesis, which asserts, in essence, that any material process can be simulated. That is, the difference between actual points in a state space of a real system and corresponding calculated points in the space of the simulated system can be made arbitrarily small by sufficient calculational effort.

There are two reasons—one practical, the other fundamental—why any actual simulation will fail when asked to perform beyond strict design limits. The practical reason has been discussed above as due to imperfect knowledge of the reality being simulated; the second, more fundamental reason, is based on Rosen's discussion of complex systems. Gödel[3] has shown that the Church-Turing Thesis fails for arithmetic. Thus, for example, it is not possible to encode the whole of arithmetic into input strings for a Turing machine in such a way that every truth of arithmetic is provable as a theorem. Rosen uses this fact as a point of departure to discuss differential equations as universal simulators.[4] He then goes on to show that "general vector field[s] cannot be described to a Turing machine, ... [and] since they cannot be encoded, they cannot be simulated. It is precisely here that Church's Thesis fails in analysis. In a precise sense, most orbits of such a general vector field are not computable." [4]

Rosen seems to be suggesting that since algorithms (computer programs) can indeed compute any computable function, and since behaviors of certain complex systems are not computable by not possessing a complete syntactic description, there can be "no independent, inherent distinction between hardware and software"[4] as the Turing machine demands. Simulations can at best repeat what "organisms *have already done*; not the things they will do."[4] A real, parallel, asynchronous neural network model is therefore necessary to emulate non-computable functions—the orbits of the

general vector field. Thus, we must progress from the neural network simulations of today to actual neural network systems of sufficient generality and power to mirror real neural activity at some level of abstraction such that they become actual synthetic organisms that learn to cope with the problems we need to solve. Only then will we have achieved our goal of creating machines with enough intelligence (or any intelligence at all, for that matter) to cope adequately with the types of problems considered here.

A Synthetic Intelligent System

To create a system exhibiting the ability to deal successfully with a complex and changing environment, a biological metaphor of an ecosystem inhabited by potentially intelligent agents is employed. The ecosystem may be changing on a continual and slow basis, as all natural systems do. A group of similar organisms presently adapted to the ecosystem is considered to be a species. It is this species that adapts to the changing environment on a genetic time scale when changes are outside a certain optimality range for the present members of that species. The individual members of the species adapt to changing conditions on a time scale determined by plasticity of the organism; this plastic period may last for the lifetime of the individual or merely during an infant and juvenile period. The important distinction is that the genetic time scale for change is much longer than the individual time scale. If changes occur too rapidly for any given individual to adapt, but not so severely as to be out of range of the available genetic pool, then a given individual may fail, but the species as a whole will adapt. If changes occur too rapidly over too extreme a range, the species becomes extinct.

Reliance on a single program or set of programs will eventually prove fatal (even if completely error free), whereas a (possibly large) set of (possibly virtual) programs can form a genetic pool, allowing mutations and crossover to bring about the evolution of a successful program. This, then, is the thrust of this work: to set up conditions in which an intelligent system may create itself through evolution.

Role of the Environment

The approved software-engineering procedure for writing a program is to start with a set of specifications that describe the desired results. Another way of viewing this process is to suppose we are constructing a function (the algorithm) that maps from a subset of internal machine states, indicated by the statements of the program, onto a subset of the possible actions that a machine can effect in the external world. The subset of this range of actions is precisely those results specified in the top-level design stages (if all has gone correctly). The programmer's job is to select the most appropriate subset in the domain of the mapping (the set of all possible machine states or statements in the programming language) that best map onto the range.

The method proposed here turns this process around and dynamically closes the loop between high-level specifications and program statements. It is this closure that is usually neglected once the initial design specifications have been made. This neglect is ultimately responsible for the brittle software systems that we are so reluctant to allow to control complex machinery. Even a conscientious effort to close this loop will not solve the problem for those systems required to function in open environments—the loop needs continuous and dynamic closure even as the environment changes.

If the domain of the function (algorithm, program) is widened to a much larger virtual space of possible mappings, the task then becomes one of evaluating the behavior of a given program instance in its range. Evaluation is generally a much simpler algorithm

and may be thought of as the inverse mapping from the range of possible actions onto the domain of virtual programs. Of course, a sufficiently generalized representative of the virtual program space must be created initially for this method to work. How this might be accomplished is discussed below.

The environment is an essential part of the ecosystem we wish to control. A specific ecosystem may consist of sensors, databases, computing engines, available software libraries, space platforms, offensive and defensive weaponry, the immune system and invading organisms. The group of functional organisms in an ecosystem act upon and react to the environment; they are in fact inseparable from the ecosystem, which is a nonlinear dynamic system of interacting parts.

The programmer/designer becomes a policy maker by providing the system with a "fitness" function that evaluates success in the environment and thus closes a loop that is normally recognized only at the system specification level in current software engineering practice. All interaction between the designer and the system is through this high-level policy algorithm, which monitors overall behavior, guiding the system to a region of local optimal functionality. Set too tight a specification, and the ensuing system loses adaptability that may be essential at some future time (e.g., when an unexpected terrain is encountered by an exploratory vehicle). Too loose a specification, and the ensuing system will behave other than desired, and may fail by default. The key point is that the closure between function and specification in a self-designing system is a continuous process.

The question of reliability assurances in the form of proofs of correctness will surely arise in the course of presenting this new (but very old) paradigm. For intelligent systems, such proofs are not only impossible, they are not even applicable: can a proof of correctness be found for the President of the United States? How, then, can we prove that a given intelligent being is going to perform correctly? The answer is that the question is ill-conceived; it is a question borrowed from one domain and forced onto another. The correct question is: Can we reasonably assume that the job will be done correctly by a certain individual? The only conceivable answer is one involving estimates and limits based on the experience of the evaluator and the candidate. The main point is that when relying on provably correct algorithms for complex, real-world situations, failure is inevitable because, sooner or later, the environment will change in an unexpected manner. With an adaptive, intelligent system, a provably correct answer may never become available in spite of our best computer science departments. On the other hand, total failure is not inevitable—the adaptive, intelligent system will quite probably muddle through to victory one way or another. Thus there is a kind of complementarity here—a too-restrictive policy measure that guarantees the existence of a correctness proof will result in brittle programs, while a more liberal policy will deny a such a proof but allow intelligent and adaptive programs to evolve.

An Optimized Retina

If we consider a retina to be a filter for complex spatial patterns that extracts certain types of information (whether in the visible spectrum or not is immaterial), then a generalized retina is a necessary part of any entity required to function in an environment that possess objects crucial to the entity's success. Pattern recognition is only one of the functions such a device must possess. Thus we look to the role of a retina as fundamental to machine perception.

Again, looking to natural systems for guidance, a retina may be specifically optimized to recognize certain features. Frog retinas responding strongly to nearby moving insects, migrating birds orienting their routes via constellations, and babies responding to

abstract human faces all come to mind as genetically designed recognition systems. Hubel[5] has shown that the human retina is also well designed for sensitivity to edges, orientation, and motion in the field of view. But such generality may not be necessary in certain applications such as identification of specific objects in a restricted environment.

A retina was constructed from a neural network based on early work in pattern recognition by Bledsoe and Browning[6] and a later elaboration by Uhr.[7] The standard n-tuple algorithm[6,7] was recast as a feed-forward neural network consisting of randomly connected feature detectors. Each feature detector has n inputs from n different retina cells (the simulated photoreceptors). In Figure 1, n is chosen to be 3, so there are 2^3, or eight, outputs of the feature detector, each of which may correspond to a feature of one or more categories that are learned by the network during a training phase. Both learning and recall involve direct access from input cells to feature detectors to the summing category nodes—no expensive relaxation process to minimum energy states[8] is necessary, nor is backpropagation[9] of errors during the learning process required—there are no graded errors in a Boolean system. Due to the statistical nature of the connectivity and the requirement that reasonable samples of each category be presented during training, the network is both fast and reasonably immune to noise—two desirable features for real-time, real-world applications.

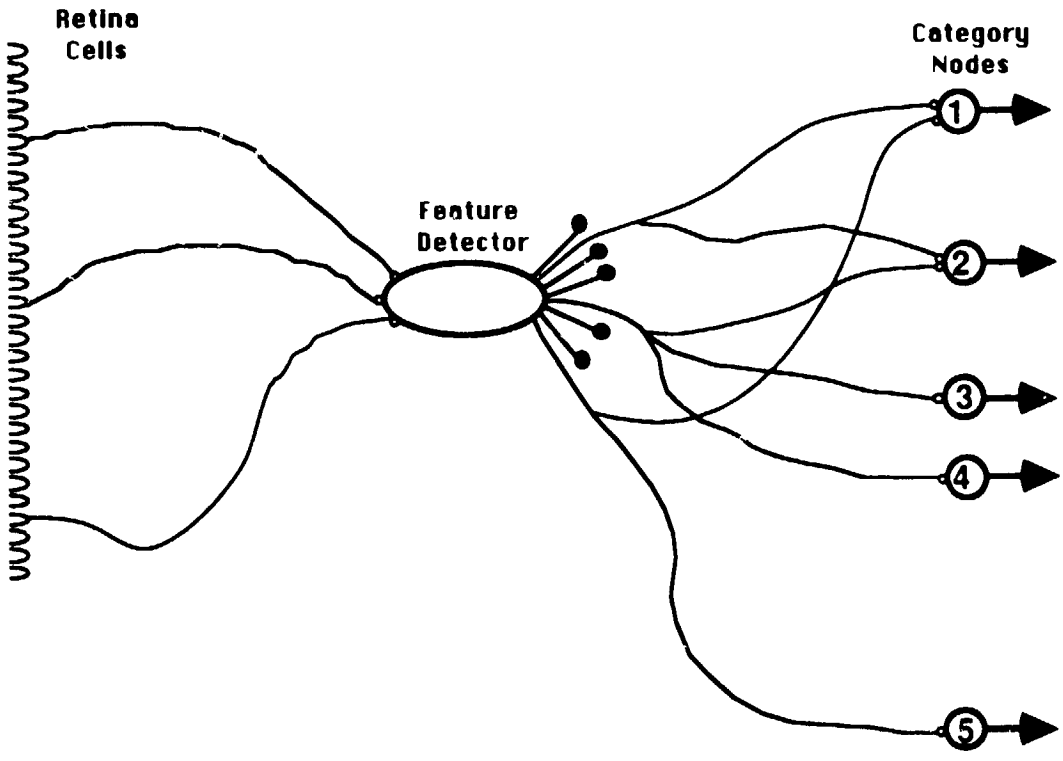

Figure 1. The feature-detector retina model is a feed-forward network activating category nodes. Activity in the retina cells is grouped into features by the m feature detectors, where m = N/n, N is the number of retina cells, and n is the order of the n-tuple. Each feature detector has a maximum of 2^n output lines connected to category nodes during training. These lines activate category nodes during recognition. Category nodes are added to the network as needed.

Both the connectivity of the network and the contents of the memory may be taken as genetic specifications. In an experiment described below, only the memory cells are subject to mutation. An alternate approach involves an evolution of feature detectors for particular sets of patterns by mutation of the connectivity between the input cells and the feature-detector nodes. In this way, invariants of the set of patterns will be encoded into the connectivity of the network. The adaptability of this type of network takes place on the evolutionary time scale—much longer than the plasticity time constants of the adaptive brain to be considered next.

An Adaptive Brain

While we can conceive of a brain without its emergent property of intelligence (indeed, examples abound), the converse of intelligence without a central nervous system (CNS) is much more difficult to imagine. The CNS used in the present work follows closely a model originally developed by Browning[10] for the Sandia Corporation. Browning chose a system modeling those biological neural networks that make use of frequency encoding of information transmitted by nerve impulses.[11] It is unclear whether frequency-coded information flow in the brain is fundamental to brain operation or whether it is merely a convenient solution to the problem of communication in a noisy environment between low-reliability components. However, recent work indicates that information is coded in the actual axon pulses and is indeed an important means of information processing, at least in certain areas of the brain.[12] Approximate coincidence of information packets traversing the network is a stringent requirement imposed by a frequency-coded network and may underlie the discrimination capabilities of the CNS. The degree of abstraction allowable in a simulated CNS is an open question—can we talk about frequency as a function of time as done by many researchers[13] or must the actual axonal spikes be simulated individually? The present work does not make the simplifying assumption of an average, differentiable frequency function, $\nu(t)$, for the neuron's output; instead, it simulates each axonal spike separately.

The model, as implemented, consists of a few hundred frequency-responsive neurons or nodes. Each node has an arbitrarily chosen number of inputs from other neurons and, on the average, a like number of outputs simulating the distribution of axonal spikes. The average connectivity is predominantly from a row of input nodes through several rows of internodes, which are not necessarily "hidden,"[14] to a row of output nodes. There is a high probability of connections in the forward direction (input to output) and a low probability in the lateral and reverse directions. The distribution function is a Rayleigh function modified by an elliptical angular distribution with the major axis aligned along the forward-backward direction.

"Synapses" are formed at the junction of the input of one node to the output of another and are modeled by a pointer associated with an input node that refers to a memory location associated with an output node. The synaptic efficacy of transmitting an axonal spike through a junction is analogous to the "weight" found in many artificial neural network models.[14] This weight is modifiable, and the modification algorithm may be altered to investigate various theories of learning and memory. To date, a simple Hebbian algorithm has been used, as well as a frequency-based version of the BCM synaptic modification.[15] Other learning theories under investigation are the drive-reinforcement model,[16] and the dual-synaptic population model.[17] Detailed comparisons of these various models of learning are not available at this time, although each of the algorithms produces reasonably satisfactory results in that the system of neurons and synapses undergoes self-organization related to the environment imposed.

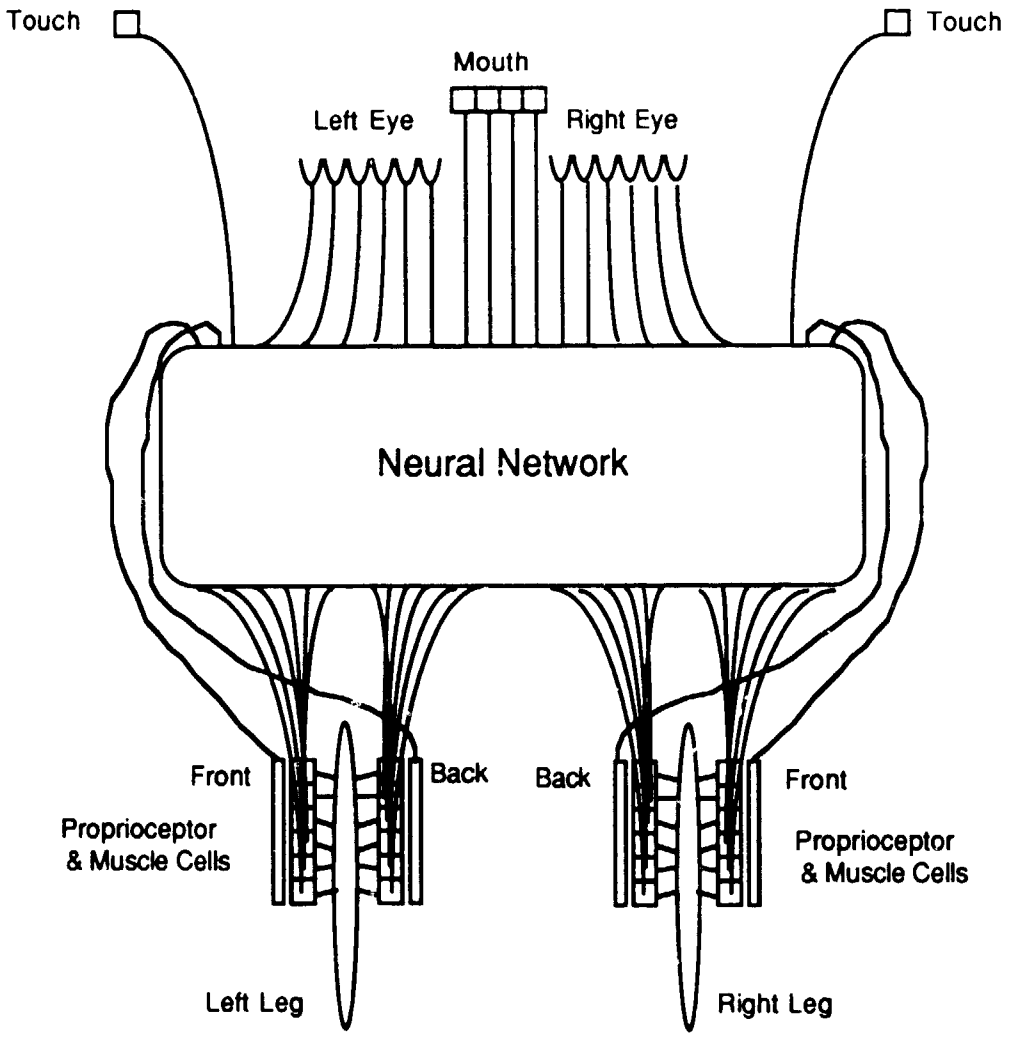

Figure 2. A synthetic organism is constructed from an adaptive, frequency-coded neural network having sensors and effectors for interacting with the environment. The organism is presented here as an insect, but the paradigm is not limited to a particular class of phenotypes.

A means of interacting with the environment was added in the form of simulated sensors (vision, taste, and touch) and effectors (groups of muscle cells). The resulting synthetic organism is shown in Figure 2. Depending on the sensors and effectors given such an organism and the environment in which it is required to function, the designer may demand anything from an artificial rat for classroom experiments in animal psychology to an autonomous vehicle required to explore the Martian surface. The biological basis of synthetic intelligence is versatile enough to produce a wide variety of successful "species." These extensions and variations are the goal of future research into electronic (and eventually, electromechanical) life forms.

Evolution and Self-Design

In a situation of sufficient complexity (as discussed above) where proofs of correctness are unattainable, the construction of infallible systems is impossible without an omniscient programmer. Darwin[18] has given us a model for the creation of optimal systems in artificial universes (independent of its

correctness in the real universe). The omniscient programmer, even if possible, is no longer needed for the creation of complex hardware and software systems when the principles of evolution are employed—a fallible program can improve its own behavior. Thus, we are on the verge of establishing the necessary conditions whereby electronic life can arise and evolve in the computer, and optimize its behavior in environments of our choosing guided by a policy of our choice. This is an extremely powerful paradigm for the design of systems to handle complex, biological-like problems, and it will lead to a revolution in the science of complex systems. Over the years, a few individuals have become interested in these ideas. One of the early investigators was W. W. Bledsoe[19], who examined some of the possibilities and problems associated with genetic models in computer science. Fedanzo[20] gave a more recent admonition to follow Darwinian precepts in problems concerning data base optimization.

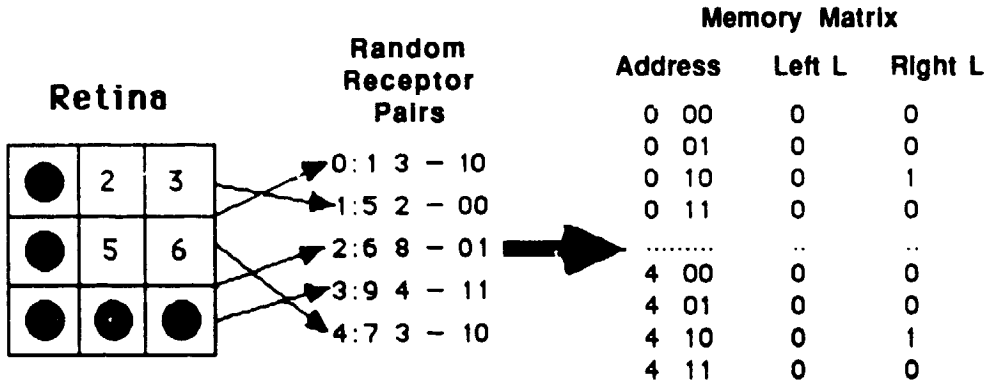

Figure 3. A classical n-tuple pattern matcher is shown for n = 2 and a 3 X 3 input array. A typical pattern is shown in the array on the left. The table in the middle shows a possible arrangement of the five possible (mostly) exclusive, randomly chosen pairs of retina cells, and the formation of subaddresses is indicated. The table on the right shows the memory matrix necessary to store two categories, one per column. The pair labeled 0 consists of the ordered retina cell pair 1,3. Cell 1 has a pixel turned on, cell 3 does not; the corresponding subaddress is therefore 10 (binary) or 2. Thus the memory matrix has an entry (if trained) at address 0, subaddress 2 in the right column, corresponding to the right-facing L-pattern.

The usual approach to adaptive systems can be made into a Darwinian approach to self-designing systems by carefully separating design and performance specifications from adaptive structures (synaptic weights, polynomial coefficients, etc.) belonging to the individual organism or, in this case, the executing computer program. One of the most noticeable differences between an adaptive genome and an adaptive system concerns time scales: adaptation in the usual sense occurs on a short time scale and within a certain program that is self-organizing in response to the environment or problem. In the case of genome evolution, the time scale is much longer, extending over many individual organisms (programs) interacting with their environment and resulting in the self-organization of the genome itself.[21]

Positive and Negative Selection

There are a wide variety of selection strategies to choose from when considering optimization based on Darwinian principles. The main idea is to introduce variations and demand that reproductive success depend on fitness (in Darwinian theory, the two concepts are synonymous). Here, we consider

the effects of the two general stragegies of positive and negative selection on fitness. Both of these strategies are amenable to simulation in a relatively simple system. The principles discussed below are well known to evolutionary biologists (e.g., Mayr[22] and Kimura[23]), and to some animal breeders, but seem relatively unknown to computer scientists.

Browning[24] suggested a simple experiment done with a data-structure version of the pattern-recognition system described above. The pattern memory is a set of binary cells addressed by patterns in the retina (see Figure 3). Mutations are made by logically complementing cell contents. The algorithm followed involves a mutation in a single, arbitrarily chosen memory cell and measuring of the success of the retina in recognizing the set of L's, both normal and reflected in the vertical. Inspection of the figure shows that there are 9 such L's possible in each orientation on a 3 X 3 grid (we are not considering reflections about the horizontal). The scores of each of the 18 L's are computed by counting 1 for each cell addressed by a particular L pattern (the L shown in Figure 3 would score 0 for the "Left" category and 2 for the "Right" category for a total score of 2). The score then becomes the "fitness" of that particular mutation.

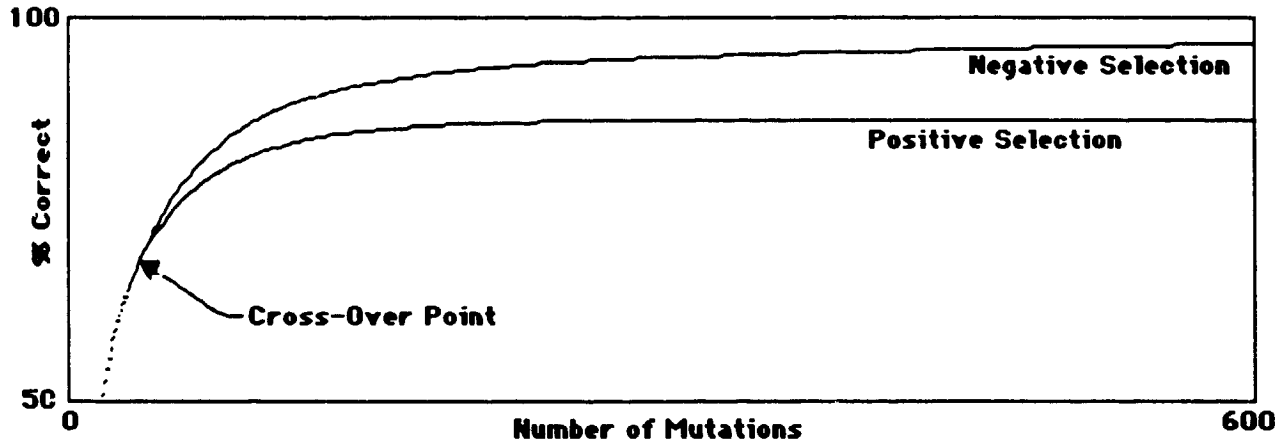

Figure 4. Results of a simple artificial genetic experiment on the memory matrix shown in Figure 2. In the 20 memory cells, 600 mutations were made for each selection strategy (see text). The strategy corresponding to negative selection clearly outperforms the positive selection case.

The experiment was run for two different selection strategies, positive and negative. "Positive" is defined as: Accept a mutation if it produced a higher score, otherwise reject it. "Negative" is defined as: Reject a mutation only if it produced a lower score. Thus we are selecting against failure, *not* for success. As Figure 4 shows, the difference is dramatic. Positive selection starts with a faster slope initially, but saturates quite early (below about 250 mutations for this 20-cell system). Negative selection, however, quickly overtakes the positive, and is still showing improvement at the 600-mutation point.

Sex and Genetic Algorithms

Employing negative selection results in improvements in the optimization algorithm chosen. Additional acceleration of the evolutionary process is possible in a sexual species where there is an opportunity for individuals to pool complementary portions of genetic material as shown by Ulam and Schrandt.[25] Application of the genetic algorithms as pioneered by Holland,[26] especially the use of cross-over methods,[27] has been shown to result in accelerated optimization for these classifier systems. There is every reason to suppose that

variants of these methods, involving a careful separation between the "phenotype" and the "genotype," would dramatically accelerate the process if applied to a collection of individuals forming a gene pool.

Results & Future Directions

One aspect of intelligent behavior is the ability to solve a problem in a surprising fashion. The system described above has generated such a surprising solution to a presumably simple problem. The problem posed to the system was to optimize the behavior of the synthetic organism by avoiding the boundaries of the environment yet keeping in motion to explore and interact with that environment. A simple fitness function set the policy by evaluating each time step of the synchronous system. The system was left to evolve on its own. The expectation was that the various parameters determining the tactile sensitivity would develop to the point where touching the boundary would initiate a sequence of network node firings effectively demanding retreat of the organism from the walls. Since the high frequency felt at the boundaries naturally proves disruptive to a frequency-coded network, as shown by previously.[28] it was natural to assume that this inherent capability would be optimized. The surprise was that this did not happen. Instead, the synthetic creature evolved—after more than four hundred generations—into a new species whose locomotion was predominantly backward. The new organisms walked backward in a very efficient manner, occasionally turning around to sense objects in the environment. There was no touch sensor on the rear, so posterior collisions with the walls had no effect on the fitness function and could not disrupt the activity of the network.

Thus, a group of parameters, or a "gene" of the system, altered to solve the problem in an efficient, surprising, and biologically reasonable manner. Indeed, one of the induced mutations discovered in *C. Elegans* (a microscopic, 850-celled worm having approximately 300 neurons) causes this very behavior.[29] Several *unc* (for uncoordinated) mutations affect the ability of the worm to move forward and backward. In particular, *unc*-4 prohibits backward motion entirely, while the *unc-7* mutation causes predominantly backward motion.[30]

As an exercise in understanding a complex system, a preliminary analysis was made of the parameter file (defining the structure and behavior of the simulated organism) before and after the evolutionary episode. The new species developed somewhere prior to 456 generations involving 1537 mutated individuals. It was assumed that a parameter residing in a group responsible for the dynamics of the muscle motion would be identifiable as responsible for the reverse locomotion (indeed, there is a parameter specifying the degree of asymmetry in the extensor-to-flexor contractions). This hope was dashed when the standard deviation of the percentage change of the parameters in the muscle-dynamics group was found to be not significantly different from that of any other of the functional groups. Indeed, the asymmetry parameter changed in the direction of increased forward impetus by about 10% rather than in the direction of reverse locomotion. The number of random trials necessary to alter the asymmetry parameter sufficiently to cause predominantly backward motion is approximately 100^5—much larger than the <1537 trials actually needed. Thus, the selectionist method of optimization is far more effective than a random method would be.

We have given an example of a system whose behavior is clearly known, but whose internal causes are not yet understood. The situation is analogous to one's pet dog: you can never understand an organism as complex as a dog, but you sure can make it sit whenever you wish (almost). Dogs have proven their reliability in complex, difficult,

and demanding situations throughout history, yet they are neither understandable (in the reductionist sense) nor provably correct.

The Future

Access to faster processors operating in parallel configurations will allow a number of additional techniques, all taken from biology, to be applied to the creation of self-designing systems. Sex, in particular, was mentioned above. Other means of accelerating evolution involve interaction between individuals of the same or different species. A process of coevolution, or "arms race" as in a mutual selection for speed in cheetahs and their prey, gazelles, is one example. Here, selection pressures force one, then the other species to excel marginally. The result is either the extinction of both or two very fast animals. Similarly, direct competition for a particular resource, such as the food objects in the simulation described in this paper, would certainly accelerate the optimization process.

All of these methods require very fast hardware and sophisticated simulation languages (for specifying ecosystems as well as neural networks). An ideal would be to let synthetic organisms interact and compete simultaneously on a set of parallel processors. Policy for coevolution between a defensive system and an offensive system would be set for victory of one "species" rather than mutual survival as in the cheetah case. Thus, an immune-system molecule could be synthetically evolved to specifically label a particular virus fragment (both in silico and in vitro).

It is anticipated that the most valuable result of this work will be a system that, upon sensing failure, will enter a mode of accelerated evolution, producing success by replaying and adapting to events that lead to the failure. This would be the ultimate adaptive system. Obvious applications are in a strategic defense system, a survey robot for nuclear power plants, and a method of responding with specific vaccines to the high rate of mutation of the AIDS virus. For NASA, there are likewise obvious applications: a planetary exploration vehicle will probably meet with failure due to unanticipated environmental effects. Any means of turning such an eventual failure into success should be welcome.

Acknowledgments

This work depended on support provided partly by the Instrumentation and Controls Division, partly by the Energy Division, both of Oak Ridge National Laboratory, and recently by the Materials Laboratory at Wright-Patterson Air Force Base.

The inspiration for this work came primarily from Dr. Iben Browning, who has been a consistent and dedicated source of ideas and perceptive and thought-provoking comments since 1986. John Peers of Novix, Inc. has provided invaluable moral and valuable technical support, as well as being instrumental in realizing a super microprocessor created by those geniuses of software and silicon, C. H. Moore and R. W. Murphy, who have my profound gratitude. Without this device, the evolutionary development described above would have been totally impractical.

References

1. Stephen Wolfram, "Complex Systems Theory," in *Emerging Syntheses in Science*, David Pines, ed., Addison-Wesley, Redwood City,1 988.
2. Robert Rosen, "Some epistemological issues in physics and biology," Cpt 22 in *Quantum Implications*, B. J. Hiley and F. David Peat, eds., Routledge & Kegan Paul, New York,1987.
3. K. Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwanter Systeme I," *Monatsheft für Mathematik und Physik*, **38**, 173-98, 1931.
4. Robert Rosen, "On the Scope of Syntactics in Mathematics and Science: The Machine Metaphor," in *Real Brains, Artificial Minds*, eds. John L. Casti and Anders Karlqvist, Elsivier Science Pub. Co., New York, 1986.
5. David H. Hubel, *Eye, Brain, and Vision*, Scientific American Library Series #22, W. H. Freeman and Company, New York, 1988.
6. W. W. Bledsoe and I. Browning, "Pattern Recognition and Reading by Machine," *1959 Proc. Eastern Joint Computer Conf.*, 225-32, 1959.

7. Leonard Uhr, Chapter 3, *Pattern Recognition, Learning, and Thought*, Prentice-Hall, Englewood Cliffs, 1973.
8. John J. Hopfield and David W. Tank, "Computing with Neural Circuits," *Science*, 625-33, 8 August, 1986.
9. P. J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavorial Sciences," *Thesis*, Harvard University, 1974.
10. Iben Browning, "A Self-Organizing System Called 'Mickey Mouse,'" Panoramic Research Report, Palo Alto, 1964.
11. John C. Eccles, *The Physiology of Nerve Cells*, The Johns Hopkins Paperbacks Ed., Baltimore, 1968.
12. Lance M. Optican and Barry J. Richmond, "Temporal Encoding of Two-Dimensional Patterns by Single Units in Primate Inferior Temporal Cortex. III. Information Theoretic Analysis," *J. Neurophysiology*, **57** (1), 162- 78, 1987. (See also parts I and II in the same issue.)
13. Robert J. Baron, *The Cerebral Computer*, Lawrence Elbaum Associates, Hillsdale, NJ, 1987.
14. J. L. McClelland and D. E. Rumelhart, Chapter 17, "A Distributed Model of Human Learning and Memory," *Parallel Distributed Processing*, James L. McClelland, *et al.*, ed., The MIT Press, Cambridge, 1986.
15. M. F. Bear, L. N Cooper, and F. F. Ebner, "A Physiological Basis for a Theory of Synapse Modification," *Science*, **237**, 42-48, 1987.
16. A. Harry Klopf, "A neuronal model of classical conditioning," *Psychobiology*, **16** (2), 85-125, 1988.
17. Gerald M. Edleman, *Neural Darwinism*, Basic Books, New York, 1987.
18. Charles Darwin, *On The Origin Of Species*, Facsimile of the First Edition of 1859, Harvard University Press, Cambridge, 1966.
19. W. W. Bledsoe, "The Use of Biological Concepts in the Analytical Study of Systems," Panoramic Research Report, Palo Alto, 1961.
20. A. J. Fedanzo, Jr., "Darwinian Evolution as a Paradigm for AI Research," *SIGART Newsletter*, **97**, 22-23, 1986.
21. W. B. Dress and J. R. Knisley, "A Darwinian Approach to Artificial Neural Systems," *Proc. 1987 IEEE Internat. Conf. on Systems, Man, and Cybernetics*, 572-77, 1987.
22. Ernst Mayr, "The Unity of the Genotype," 71-72, in *Genes, Organisms, Populations: Controversies over the Units of Selection*, Robert N. Brandon and Richard M. Burian, eds., The MIT Press, Cambridge, 1986.
23. Motoo Kimura, *The Neutral Theory of Molecular Evolution*, Cambridge University Press, Cambridge, 1985.
24. Iben Browning, personal communication, 1987.
25. S. Ulam and R. Schrandt, "Some Elementary Attempts at Numerical Modeling of Problems Concerning Rates of Evolutionary Processes," *Physica* **22D**, 4-12, 1986.
26. J. H. Holland, "A Mathematical Framework for Studying Learning in Classifier Systems," *Physica* **22D**, 307-17, 1986.
27. David E. Goldberg, *Genetic Algorithms*, Addison-Welsey, Reading, MA, 1989.
28. W. B. Dress, "Frequency-Coded Artificial Neural Networks: An Approach to Self-Organizing Systems," *Proc. IEEE First Annual Internat. Conf. on Neural Networks*, Vol II, 47 - 54, 1987.
29. Tom Coohill, Western Kentucky University, personal communication, 1988.
30. Jonathan Hodgkin, *Gene List for the Book of the Worm*, MRC Laboratory of Molecular Biology, Cambridge, England, 1988.

DISCLAIMER